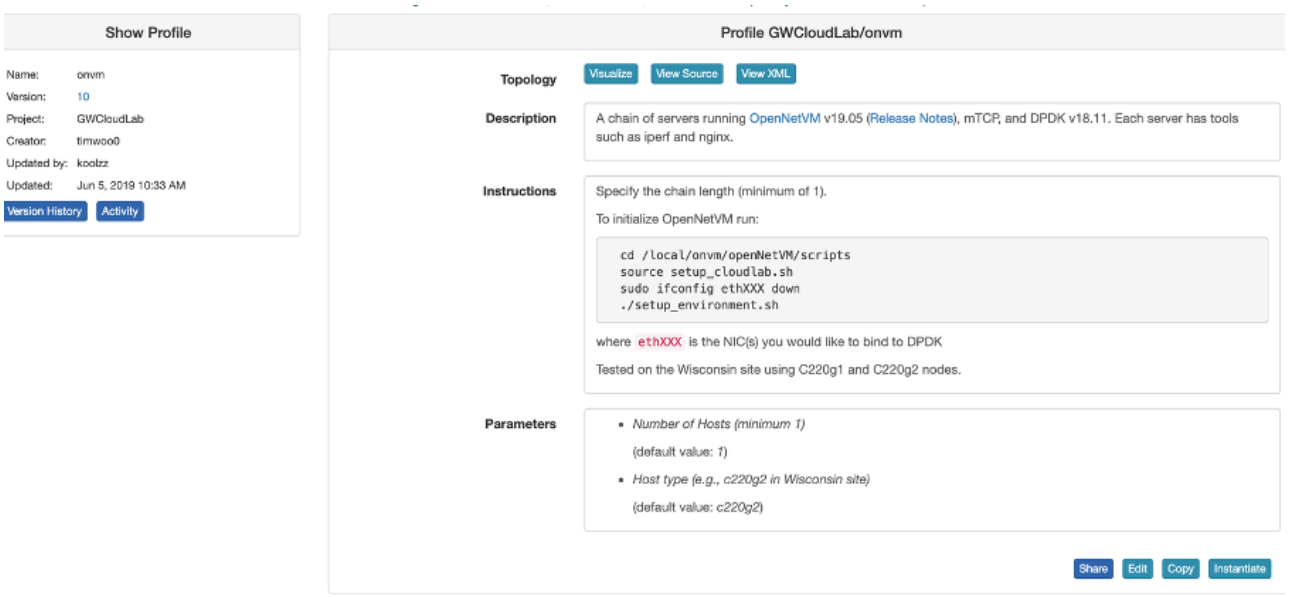
ONVM Documentation

Getting Started

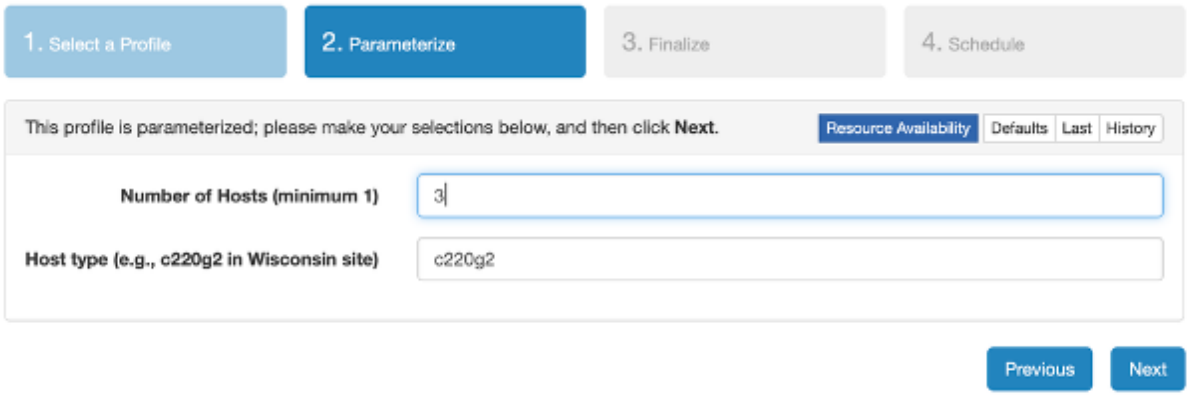
- This tutorial assumes that you have access to CloudLab and a basic working knowledge of CloudLab and SSH
- Also ensure that you have followed the development environment configuration instructions in our Dev Environment Wiki page
    - Specifically, make sure that you follow the steps in “Getting Started”, “Setup OpenSSH”, and everything up to instantiating an experiment in the “CloudLab Work Environment” section

Instantiating the ONVM CloudLab Profile

- Follow the above link to the ONVM profile on CloudLab
- Click “Instantiate”



- Enter the number of hosts you want — for a three-node topology, enter “3”
- Ensure that the host type is “c220g2”



- Click “Next”
  - The generated topology image should somewhat resemble the image below

- Optionally, enter a name for the experiment in the “Name” field
- Click “Next”
- Click “Finish”
- Wait for the experiment to boot up

Connecting to CloudLab in Visual Studio Code via SSH

- Click “List View” to see the SSH commands to connect to your nodes

- Ensure that your generated SSH command works by running it in terminal

For development within the Visual Studio Code environment:

- See more detailed setup instructions in our Dev Environment Wiki if you wish to use the VS Code environment for your setup

The following steps should be performed **for each node:**

- Copy relevant information into your `~/.ssh/config` file:

```
Host NodeXAddress
  HostName NodeXAddress
  Port 22
  User CloudLabUsername
  IdentityFile ~/.ssh/PrivateKeyFile
  AddKeysToAgent yes
```

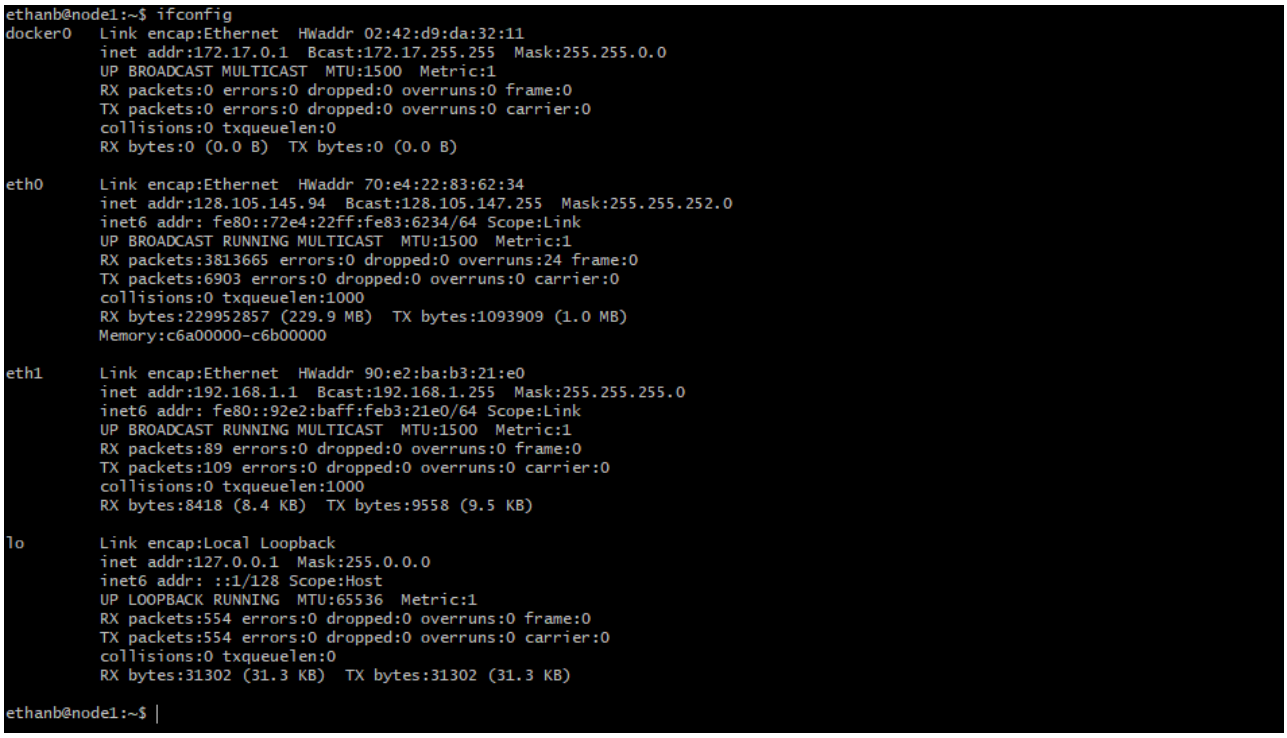
  - Note that you can add other options as necessary
- Open Visual Studio Code
- Click the green Remote-SSH extension button (SSH logo) in the bottom-left corner
- Select `Remote-SSH: Connect to Host` from the options that appear in the command palette
- Select the address of the node you want to connect to
- Visual Studio Code will automatically connect and set itself up
  - See Troubleshooting Tips for connection issues and Fixing SSH File Permissions for permissions errors
- Once connected, navigate to the openNetVM repository folder: `cd /local/onvm/openNetVM`
- Now, finish configuring your workspace by selecting **File → Open** or **File → Workspace** and selecting the openNetVM folder (`/local/onvm/openNetVM`)

Setting Up a Three-Node Topology

The goal of this document is to configure the three nodes so that the first can act as a client, the third as a server, and the second node will act as a middlebox running OpenNetVM. The first and third nodes will use the kernel network stack, while the second will use DPDK.

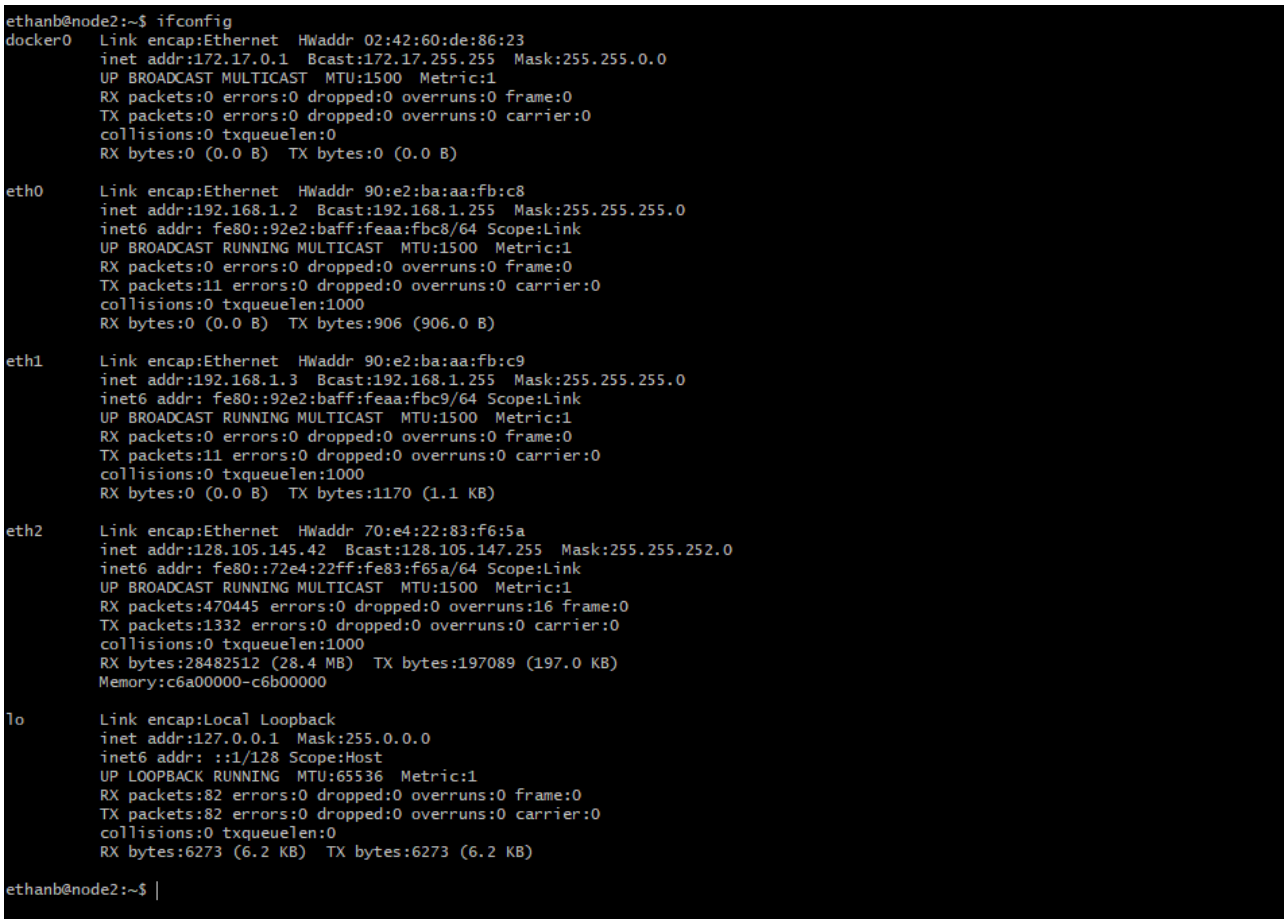
Ensuring That Nodes Are Connected

- Connect to your CloudLab nodes in either Visual Studio Code or any SSH client
- With a three-node topology, your first node (node1) should be connected to one port in your second node (node2) and your third node (node3) should be connected to the other port in your second node (node2). Notice that this forms a “chain-like” structure like the one visualized in the topology image generated by CloudLab
- To determine which NICs are connected on each node, SSH into the node and run `ifconfig`



Output of `ifconfig` on node1

  - The connected NIC is the one with the local IP subnet. For the first node, it should be `192.168.1.1`
    - Note that the local subnet is `192.168.1.x`. This means that each of the NICs should have their `inet addr` field in the `ifconfig` command output start with `192.168.1.`.
    - For each NIC in the connection chain, the IP address should be `192.168.1.<previous + 1>`. This means that the first should be `192.168.1.1`, the second should be `192.168.1.2`, and so on. Note that since node2 (and any other intermediate nodes in the case of a chain with more than three nodes) has two NICs configured for this, it will have two NICs with local addresses. This is seen in the below screenshot.



Output of `ifconfig` on node2

- The NIC names and ports (eg eth0 or eth1) can be completely random, but always have the local IP address mask (start with `192.168.1`)

Bind Intermediate Nodes to DPDK

Before running the ONVM manager, we need to ensure that the connected NICs on node2 are bound to DPDK. DPDK has a script to determine whether NICs are bound or not.

- Identify which NICs are connected to the other nodes using `ifconfig` on node2 and checking the `inet addr` against the expected output above
- Navigate to the openNetVM folder that comes pre-installed on each node using `cd /local/onvm/openNetVM`
- Pull the most recent version of openNetVM from GitHub: `git pull upstream master`
- Unbind the connected NICs: `sudo ifconfig ethxxx down`
- Run the ONVM `setup_environment.sh` script
  - `cd scripts`
  - `source ./setup_cloudlab.sh`

- `./setup_environment.sh`

```
ethanb@node2:/local/onvm/openNetVM/scripts$ ./setup_environment.sh
Setting up hugepages
Removing currently reserved hugepages
Unmounting /mnt/huge and removing directory
Reserving hugepages
Creating /mnt/huge and mounting as hugetlbfs
Huge pages successfully configured
igb_uio 13476 0 - Live 0x0000000000000000 (OX)
IGB UIO module already loaded.
Checking NIC status

Network devices using kernel driver
===================================
0000:01:00.0 'I350 Gigabit Network Connection 1521' if=eth2 drv=igb unused=igb_uio *Active*
0000:01:00.1 'I350 Gigabit Network Connection 1521' if=eth3 drv=igb unused=igb_uio
0000:06:00.0 '82599ES 10-Gigabit SFI/SFP+ Network Connection 10fb' if=eth0 drv=ixgbe unused=igb_uio
0000:06:00.1 '82599ES 10-Gigabit SFI/SFP+ Network Connection 10fb' if=eth1 drv=ixgbe unused=igb_uio

No 'Crypto' devices detected
============================

No 'Eventdev' devices detected
==============================

No 'Mempool' devices detected
=============================

No 'Compress' devices detected
==============================
Binding NIC status
Bind interface 0000:06:00.0 to DPDK? [y/N] y
Binding 0000:06:00.0 to dpdk
Bind interface 0000:06:00.1 to DPDK? [y/N] y
Binding 0000:06:00.1 to dpdk
Finished Binding

Network devices using DPDK-compatible driver
============================================
0000:06:00.0 '82599ES 10-Gigabit SFI/SFP+ Network Connection 10fb' drv=igb_uio unused=
0000:06:00.1 '82599ES 10-Gigabit SFI/SFP+ Network Connection 10fb' drv=igb_uio unused=

Network devices using kernel driver
===================================
0000:01:00.0 'I350 Gigabit Network Connection 1521' if=eth2 drv=igb unused=igb_uio *Active*
0000:01:00.1 'I350 Gigabit Network Connection 1521' if=eth3 drv=igb unused=igb_uio

No 'Crypto' devices detected
============================

No 'Eventdev' devices detected
==============================

No 'Mempool' devices detected
=============================

No 'Compress' devices detected
==============================
Disabling hyperthreading...
CPU(s):              40
Thread(s) per core:  1
Core(s) per socket:  10
Socket(s):           2
Environment setup complete.
ethanb@node2:/local/onvm/openNetVM/scripts$
```

Output of `./setup_environment.sh`

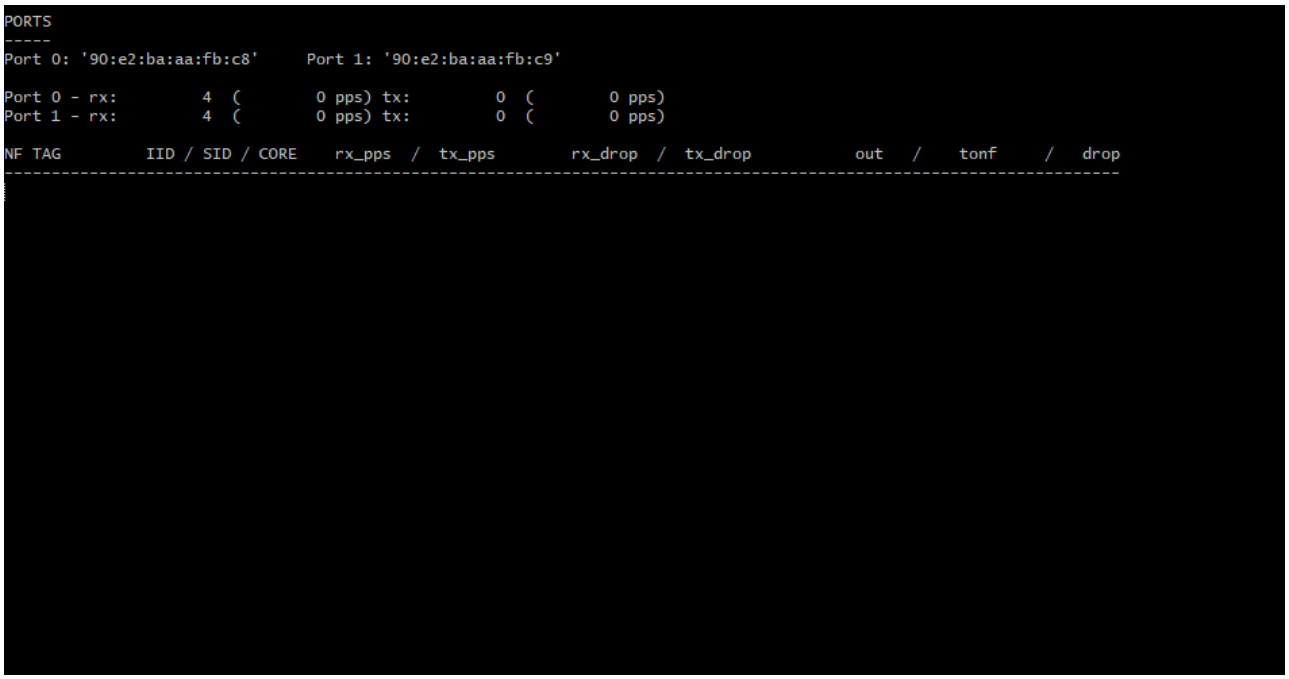
- Ensure that you see the **two** NICs in the “Network devices using DPDK-compatible driver”
  - If you only see one NIC, it’s possible that you did not unbind the other NIC from the kernel driver using `sudo ifconfig ethxxx down`. Instructions for that are above.

Verifying Node Chain Connections with openNetVM

Run the openNetVM Manager and Bridge NF

In the case of the three-node topology, we only need to run openNetVM on node2. These instructions should only be performed on all intermediate nodes in a longer chain.

- Navigate to the openNetVM folder: `cd /local/onvm/openNetVM`
- Compile the Manager: `cd onvm && make && cd ..`
- Compile the NFs: `cd examples && make && cd ..`
- Run the Manager: `./onvm/go.sh 0,1,2 3 0xF8 -s stdout`
  - The manager should show both ports running



- In another terminal pane, run the Bridge NF
  - `cd examples/bridge`
  - `./go.sh 1 1`

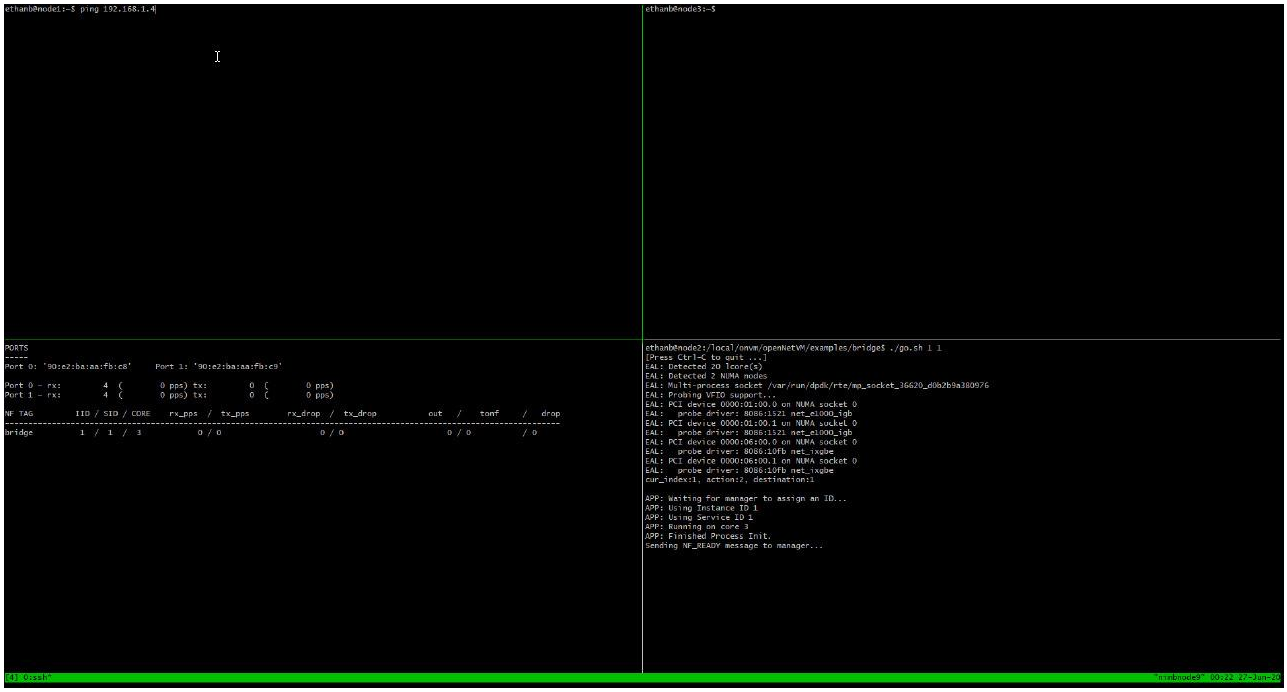
```
ethanb@node2:/local/onvm/openNetVM/examples/bridge$ ./go.sh 1 1
[Press Ctrl-C to quit ...]
EAL: Detected 20 lcore(s)
EAL: Detected 2 NUMA nodes
EAL: Multi-process socket /var/run/dpdk/rte/mp_socket_36578_d0a7b3d9f54a6
EAL: Probing VFIO support...
EAL: PCI device 0000:01:00.0 on NUMA socket 0
EAL:   probe driver: 8086:1521 net_e1000_igb
EAL: PCI device 0000:01:00.1 on NUMA socket 0
EAL:   probe driver: 8086:1521 net_e1000_igb
EAL: PCI device 0000:06:00.0 on NUMA socket 0
EAL:   probe driver: 8086:10fb net_ixgbe
EAL: PCI device 0000:06:00.1 on NUMA socket 0
EAL:   probe driver: 8086:10fb net_ixgbe
cur_index:1, action:2, destination:1

APP: Waiting for manager to assign an ID...
APP: Using Instance ID 2
APP: Using Service ID 1
APP: Running on core 3
APP: Finished Process Init.
Sending NF_READY message to manager...
```

Ping Between Nodes in Chain

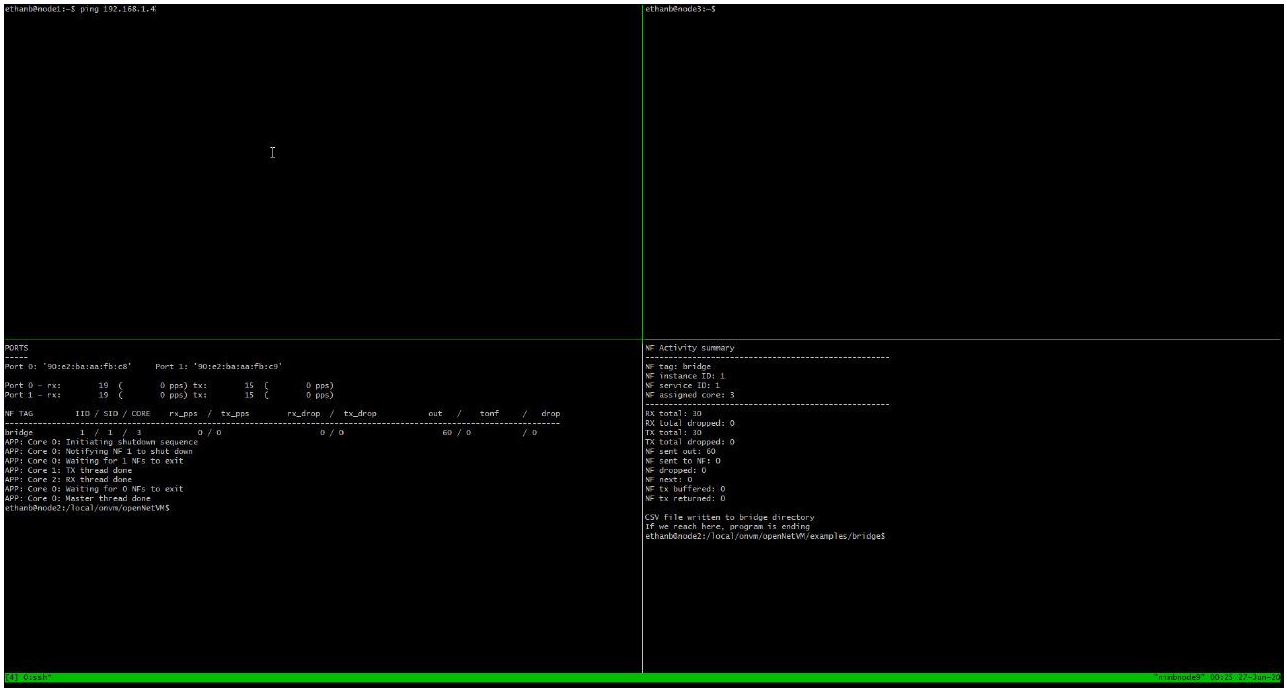
When the ONVM Manager and Bridge NF are running, we can ping from node1 to node3, using node3's local IP address, despite node1 and node3 not being directly connected. We can also ping node1 from node3 using node1's local IP address. The following steps can be performed on either node1 or node3. Just ensure that you are using the opposite node's direct IP address. The direct IP of node1 should be `192.168.1.1` and the direct IP of node3 should be `192.168.1.4`. Since these are not bound to DPDK, we can still verify this by doing `ipconfig` on either node.

- Ping the opposite node: `ping 192.168.1.x` where `x` is the node's NIC number in the chain. You will see the number of packets sent updated in the manager



Output from pinging from node1 to node3

- Note that there is no output in node3. You can verify that openNetVM is enabling the connections by closing the Manager and/or Bridge NF and repeating the ping command



Output from pinging node1 to node3 when ONVM Manager is offline